A Low Latency, Loss Tolerant Architecture and Protocol for Wide Area Group Communication[*]

Yair Amir, Claudiu Danilov, Jonathan Stanton[†]
Department of Computer Science
Johns Hopkins University
3400 North Charles St.
Baltimore, MD 21218 USA
{yairamir, claudiu, jonathan}@cs.jhu.edu

Abstract

Group communication systems are proven tools upon which to build fault-tolerant systems. As the demands for fault-tolerance increase and more applications require reliable distributed computing over wide area networks, wide area group communication systems are becoming very useful. However, building a wide area group communication system is a challenge. This paper presents the design of the transport protocols of the Spread wide area group communication system. We focus on two aspects of the system. First, the value of using overlay networks for application level group communication services. Second, the requirements and design of effective low latency link protocols used to construct wide area group communication. We support our claims with the results of live experiments conducted over the Internet.

Keywords—**Group Communication, Overlay Networks, Reliable Multicast, Wide Area Networks, TCP/IP.**

1 Introduction

There exist some fundamental difficulties with high-performance group communication over wide-area networks. These difficulties include:

- The characteristics (loss rates, amount of buffering) and performance (latency, bandwidth) vary widely in different parts of the network.
- The packet loss rates and latencies are significantly higher and more variable than on local area networks.
- It is not as easy to implement efficient reliability and ordering on top of the available wide area multicast mechanisms as it is on top of local area hardware broadcast and multicast. Moreover, the available best effort wide area multicast mechanisms come with significant limitations.

More and more applications are starting to use group communication systems to enable fault-tolerance and high-availability or to create large scale complex interactive applications. In high-availability application domains such as stock markets and command and control systems, group communication systems have been used for years. With the increased demand for high reliability in standard networked business services such as web servers and databases, these uses are becoming more widespread.

A more recent area of interest in group communications is wide-area distributed simulation and distributed object applications that require a large number of active objects which keep state and communicate with low latency requirements. The requirement for many active objects often necessitates an equally large number of active groups in the system. Other wide-area applications that could benefit from group communication are database replication, network and service monitoring, and collaborative design and interaction.

Most existing group communication systems are limited in their ability to provide wide area services with low latency because their protocols were designed with local area networks in mind. Because of these limitations, most applications today either use a TCP based client-server architecture sending one copy of the data to each recipient or use IP-Multicast for data dissemination coupled with external reliability and ordering servers. Each of these choices has some limitations either in performance/scalability issues or in complexity and ease of use.

[*]This work was supported in part by grants from the National Security Agency (NSA) and the Defense Advanced Research Projects Agency (DARPA).

[†]This is the Tech Report version. Cite as TR 00/99 -xx

We address these problems by creating a group communication service, called Spread, which provides high performance in both local area and wide area networks. Spread provides all the services of traditional group communication systems, including unreliable and reliable delivery, FIFO, causal, and total ordering, and membership services with strong semantics. Two advances allow this: the incorporation of an overlay network architecture in the group communication system, and the development of a new point-to-point protocol for the wide area network, tailored to that architecture.

Spread creates an overlay network that can impose any arbitrary network configuration including for example, point-to-multi-point, trees, rings, trees-with-subgroups and any combinations of them to adapt the system to different networking environments. Coupled with that, a new point-to-point protocol for the wide area network, the Hop protocol, is designed for this environment. The Spread architecture allows multiple protocols to be used on links between sites and within a site. To validate the usefulness of the Hop protocol for this environment, we compare it with using TCP on the wide area links between sites.

By targeting Spread at wide area networks we did not compromise performance over local area networks. Spread achieves similar performance to the best existing group communication systems in local area networks[1] with somewhat higher cpu requirements to achieve the same performance. Spread is also available in a separate version specifically tuned for local area networks which has no additional requirements.

Spread is very useful for applications that need the traditional group communication services such as causal and total ordering, and membership and delivery guarantees, but also need to run over wide area networks. In fact, it is the first available group communication system to fully support strong semantics across wide area networks as far as we know. In addition, other applications may find Spread a better fit compared with the different reliable IP-Multicast schemes because of several technical differences:

- *Scalability with the number of collaboration sessions.* IP-Multicast is very good at supporting a small number of sessions, each broadcast to a large number of clients. Spread, on the other hand, can support a large number of different collaboration sessions, each of which spans the Internet but has only a small number of participants. The reason is that Spread utilizes *unicast* messages on the wide area network, routing them between Spread nodes on the overlay network. Therefore, IP-Multicast related resources are not required on the network routers.

- *Scalability with the number of groups.* Spread can scale well with the number of groups used by the application without imposing any overhead on network routers. Group naming and addressing is no longer a shared resource (the IP address for multicast) but rather a large space of strings which is unique per collaboration session.

- *Routing.* All of the current IP-Multicast routing methods build routing trees in an incremental way. This is very good for being able to scale to millions of users on a session. However, since Spread has to maintain membership, it requires little additional work to reconstruct routing trees every time the membership changes. This provides Spread with the ability to construct optimal routing trees. Note, though, that these trees are on the overlay network and not on the physical network. Both IP-Multicast and Spread support pruning.

The Spread toolkit is available publicly. An early version of the system is used by several other organizations for research and practical projects. The toolkit supports cross-platform applications and has been ported to several Unix platforms as well as Windows and Java environments. More details on the Spread system can be found at http://www.spread.org/ along with a white paper and programming documentation.

2 An Overlay Network Architecture for Wide Area Group Communication

Our goal for a multicast architecture is to facilitate efficient group communication services for local and wide area networks. These services include unreliable and reliable dissemination of messages to process groups, ordering guarantees on messages, and membership services. These services usually adhere to strict semantics such as Virtual Synchrony[6] and its flavors[16, 8]. This range of services can be used to more easily develop, or make fault-tolerant, applications ranging from replicated database servers to group collaboration tools to streaming multimedia.

The Spread system uses generally long-running daemons to establish the basic message dissemination network and provide basic membership and ordering services, while user applications link with a small client library, can reside anywhere in the network, and will connect to the closest daemon to gain access to the group communication services. There is a small cost to using a daemon-client architecture, which is extra context-switches and inter-process communication, however, on modern systems this cost is minimal in comparison with wide-area latencies.

A "site" in Spread consists of a collection of daemons which can all communicate over a broadcast or multicast domain. This is usually limited to a local area network. We will use the term “site” to refer to this collection of locally connected daemons as a whole. Each site selects one daemon, based on the current membership of the site, that acts as gateway, connecting all the members of the site to other sites.

The Spread group communication system architecture solves five main problems: end-to-end reliability, configuration of an overlay network, low-latency forwarding over high-latency links, the high cost of membership changes, and scalability.

2.1 End-to-End Reliability

Guaranteed end-to-end reliability is a result of the properties of two separate protocols in Spread. First, each link guarantees that all packets sent on it will eventually be reliably received on the other side as long as there is not a membership change. Second, the membership protocols detect lost links, crashed daemons, and network partitions and then recover the necessary state to continue making progress and to report to the application if any messages were not able to be reliably delivered because of crashes or partitions. The three possible cases are:

- *No failures, no slow receivers.* Here the eventual reliability of each link is sufficient to result in all messages getting to all recipients reliably.

- *No failures, slow receivers.* Here we impose a global window of outstanding messages which will limit the speed at which new messages are generated to the rate the slowest receiver can handle. Combined with link reliability, this produces end-to-end reliability.

- *Failures.* Here the membership protocol takes over and recovers the system state, resends messages as needed, and informs the application of which messages have guaranteed reliability according to the semantics defined by Extended Virtual Synchrony Model [16].

In general, Spread decouples the dissemination and local reliability mechanisms from the global ordering and stability protocols. This decoupling allows messages to be forwarded on the network immediately despite losses or ordering requirements. The only place where messages are delayed by Spread is just before delivering them to the clients, to preserve the semantic guarantees. Decoupling local and global protocols also permits pruning, where data messages are only sent to the minimal necessary set of network components, without compromising the strong semantic guarantees.

Spread allows different low level protocols to be used to provide reliable dissemination of messages, depending on the configuration of the underlying network. Each protocol can have different tuning parameters applied to different portions of the network. In particular, Spread integrates three low-level protocols: one for local area networks called Ring, and two for the wide area overlay network connecting the local area networks: the standard TCP, and our new protocol called Hop.

2.2 Overlay Networks

We define “overlay network” as a virtual network constructed such that each link connects two edge nodes in an underlying physical network, such as the Internet. Each virtual link in the overlay network can translate into several hops in the underlying network, and the cost attached to a virtual link is some aggregated cost of the underlying network links over which it travels.

Spread constructs an overlay network between all the sites that have currently active daemons. This network is constructed based on information contained in the static configuration file given to Spread, the current daemon membership list, and any available network cost information. These sources produce a network that dynamically changes as daemons are started or crash, and as partitions and merges occur. The configuration file provides information about all potential machines in the Spread system, but does not constrain which members are currently running.

The overlay network is used to calculate source based optimal routing paths from each source to all other Spread daemons. Source based routing over the shortest path[17] produces the best routes when it is feasible to do. For the application domain Spread is targeted at, the calculations associated with source based routing are feasible. After a membership change a new overlay network is constructed and the routing trees are recalculated based on the new network.

The membership service provides each daemon with an identical view of the current global membership and the link weights of the current overlay network. Then, each Spread daemon independently calculates a shortest path multicast tree from each site to every other currently connected site. Since each daemon uses identical weights and an identical graph they are guaranteed to compute the same routing trees. The routing calculation uses Floyd-Warshall[9] all-pairs shortest path algorithm.

When configuring Spread, one can set weights for every potential link in every direction. Thus, different overlay networks can be constructed based on the needs of the application and knowledge about the behavior of the underlying network. Actually, since different weights can be set for every link in both directions, the flexibility exists to create most practical networks. Figure 1 presents a sample set of sites located around the United States and the over-

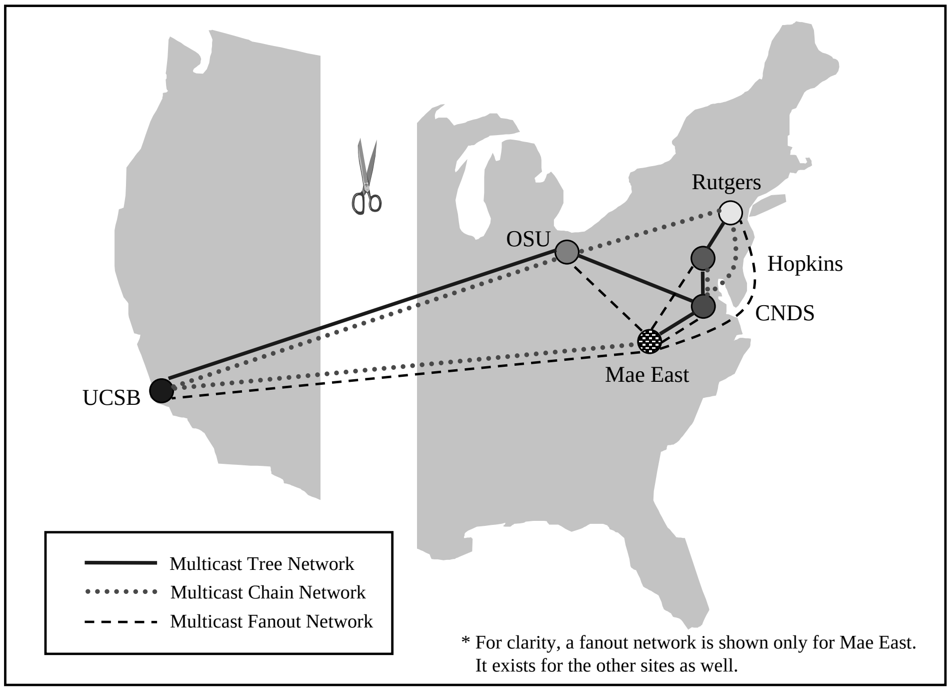

Figure 1. Network Testbed

lay networks that were constructed between them based on different needs. Each site may contain several tens of daemons, each of them serving many clients. The figure shows the real network used for the experiments reported in Section 4. The overlay network labeled “Multicast Tree Network” shows a shared tree network where the tree rooted at each site is the same. The network labeled “Multicast Fanout Network” shows a point-to-multipoint multicast network where each site has direct connections to all other sites (only the tree rooted at Mae East is shown for clarity reasons).

As mentioned before, when changes in the membership occur, such as daemons dying or starting up, the Spread membership service automatically discovers all live daemons and creates a new membership including all of them. Then a new overlay network is constructed connecting the live daemons using minimal cost links. For these experiments the overlay network routing configuration was hardwired to the described configurations for the purpose of evaluating the behavior of the link protocols on these specific networks. We were not trying in this paper to evaluate the dynamic reconfiguration algorithms.

The major cost of using an overlay network is that since the overlay is constructed only between end nodes in the underlying network, inefficiencies exist in the routing paths. However, this disadvantage is outweighed by several key benefits the overlay architecture provides: First, the algorithms used in the overlay network can be easily changed and do not require changes to basic network infrastructure (e.g. routers). Second, routers can be made simpler and faster, while complex protocols and processing can occur on end nodes where more abundant resources exist. For example, as a result of the difficulties encounterd while deploying and upgrading IP-Multicast in routers, most of the work on high level multicast services, such as reliability, has used an overlay network approach.

2.3 Low-latency Forwarding

Group communication applications are often very latency sensitive, both the low-level membership and ordering protocols themselves, and the high-level applications built on top of a group communication toolkit. Spread uses two approaches to solve this problem. First, the global ordering protocols used in Spread were designed to be latency-insensitive as much as possible. Second as many sources of added latency in the dissemination path (for both data messages and control messages) were removed.

The most significant change was to design a point-to-point reliability protocol which did not delay packets until they were in order, but rather could forward them out of order and let a higher level protocol deal with delivering them to the application in order. In this way, messages are delayed only at the receiver and not while in transit. Therefore, no latency is added because of reliability and routing. Latency is incurred only due to flow control on the network and to preserve semantics on the receiver. This protocol is discussed in depth in Section 3.2.

2.4 Membership Costs

Spread uses a daemon-client architecture. This architecture has many benefits, the most important for the wide-area setting being the resultant ability to pay the minimum necessary price for different causes of group membership changes. Simple joins and leaves of processes translate into a single message. A daemon disconnection or connection does not pay the heavy cost involved in changing wide area routes. Only network partitions between different local area components of the network require the heavy cost of a full-fledged membership change. Luckily, there is a strong inverse relationship between the frequency of these events and their cost in a practical system. The process and daemon membership correspond to the more common model of "Lightweight Groups" and "Heavyweight Groups"[12].

Using the Spread architecture, one can envisage millions of Spread overlay networks concurrently sharing the Internet. Each overlay network is formed on behalf of one application, which might support tens of users collaborating on a project. In contrast, such a scenario cannot be envisaged using IP-Multicast because of the scalability limits of routers in handling millions of concurrent groups.

Core based trees alleviate some of the scalability problems with IP-Multicast groups, however, they do not remove the limits on the number of concurrent groups because the same core backbone routers will end up being used by millions of groups which cross them. The key difference is that each daemon handles only the groups associated with its application, and needs no knowledge of any groups used by other Spread daemon overlay networks.

Of course, there are some applications that are only possible in a model such as IP-Multicast. For example, large scale multimedia broadcasts of a limited number of channels to millions of users is only possible with router support.

Although, a wide area membership service is operational in Spread (which is available for download), the details of a wide area membership service is beyond the scope of this paper.

2.5 Scalability

A group communication system always involves trade-offs in performance, scalability, and services. As mentioned above, Spread supports a very large number of groups active at any time(such as several thousand), and a very large number of active Spread configurations at any time (essentally unlimited). Spread also uses a hierarchal architecture to scale the number of daemons and users into the tens to hundreds of daemons and thousands of users. The hierarchy Spread uses consists of three levels:

- *Sites.* There are between 1 and 100 sites. Each site is connected to some of the others through point-to-point WAN links forming a multicast forest (optimal source-based trees).
- *Daemons.* Each site can contain between 1 and 50 daemons. The site uses a ring based protocol to provide ordering, reliability, and flow control amoung the daemons.
- *Users.* Each daemon can support (in theory) between 1 and a few thousand users connected at once (in practice we have tested upto a couple hundred). The daemon accepts TCP or Unix Domain Socket connections from client applications.

3 Wide Area Link Protocols

Given the framework above, each link between two daemons that are directly connected on the overlay network, can be formed by one of several protocols. In this section we discuss the requirements for a wide area link protocol and describe the Hop protocol that addresses these requirements.

Before we dive into wide area protocols, it is worthwhile to explain what protocol is used within each 'site'. Within a site, the Spread daemons use the Ring protocol to provide data dissemination, reliability, and flow control. The Ring protocol uses the unreliable multicast service provided by IP, and is based on the same ideas as the Totem[4] system and achieves similar performance. Each site can include several tens of daemons who all participate in the Ring protocol, and who are treated as one entity on the wide-area network.

The Ring protocol is based on passing a token around the the members of the site, with each member sending new multicast messages when it holds the token. When the time to pass the token is very small this can work quite well, however in high latency wide-area networks, the time during which no member can send because the token is in transit becomes very significant. Additionally, since in a wide-area network the network medium is not shared as it is in a LAN, it is possible for multiple daemons to be sending at the same time which token passing will not allow. Finally, the Ring protocol becomes more fragile when the chance of packet loss is non-negligible as losing the packet is costly to recover from.

Spread couples these rings with a wide area protocol such as the Hop protocol in order to create a complete optimized network connecting a set of local area networks.

3.1 Requirements for Wide Area Link Protocols

For wide-area links, two protocols are discussed in this paper: TCP and the Hop protocol. Here we will compare TCP as the link protocol in a Spread created multicast tree, with the use of the Hop protocol in a Spread multicast tree. We will call these two protocol choices: TCP based multicast, and Hop based multicast. To provide the global group communication services, the link protocols must provide eventually reliable transport and link-level flow control.

TCP is a very mature protocol that provides both reliable transport and flow control. However, TCP also provides other guarantees, in particular FIFO ordering. TCP will not deliver any data out of order, which can cause problems when multiple TCP connections are chained to create our overlay network. Specifically, since messages are delayed until they can be delivered in order, losses will cause delays in forwarding the data to the next link down the tree. Furthermore, when the data is finally recovered, a burst of buffered data is then immediately forwarded down the tree. These micro-delays and micro-bursts cause end-to-end latency increases and an increased burstiness on the network which itself causes degraded performance.

These issues call for a design of a protocol which better fits the requirements of a wide area link protocol in our overlay architecture.

The Hop protocol is designed to provide only the required services, specifically reliability and flow control. The Hop protocol forwards packets as soon as they are received even if prior packets are missing. The Spread system provides all the standard group communication orderings at a higher level, where messages are only delayed just before being delivered to the application.

Both the TCP based multicast and Hop based multicast protocols have several advantages over emulated multicast, where end-to-end links are used between all the application instances. Not only can they utilize multicast trees to avoid sending N copies of the data across the sender's network, but they can also achieve localized recovery of lost packets without requiring the original sender to re-send the data. Localized recovery is crucial for high latency multicast networks, not only for large[13], but also for small groups where the members are widely dispersed in the network. Each unicast link in the Spread multicast tree has buffers on the sending side to store data until it has successfully reached the other end of the link.

Most current reliable multicast protocols have some form of localized recovery, such as creating virtual local subgroups along the tree[13], or using nack avoidance algorithms and expanding ring nacks[10]. All of these techniques create an approximation of recovering the missing data from the closest node. Spread has an accurate knowledge of the current membership and the structure of the overlay network. As discussed in Section 2.1 we do not have one protocol provide end-to-end reliability directly. Instead, we rely on link reliability, liveness and global flow control to guarantee end-to-end reliability. Each link is guaranteed to eventually transfer each packet to the other side. The source dissemination tree from the sender to each receiver guarantees that each daemon will eventually receive all the necessary data from its parent. Failure recovery and liveness is guaranteed by Spread's membership protocols.

Before we discuss the Hop protocol, we mention that as a global optimization to allow packing of small messages and fragmentation of large messages, all of the network protocols used by Spread (Ring, Hop, TCP) actually operate on packets constructed of one or more data fragments and control messages. All control information used by the protocols is piggybacked on data packets when possible. If no data is available, control messages are sent as separate packets.

3.2 The Hop Protocol

The Hop protocol operates over an unreliable datagram service such as UDP/IP. The core goal of the Hop protocol is to provide the lowest latency and highest throughput possible when transferring packets across wide-area networks. The key elements of the Hop protocol are:

- *Non-Blocking:* packets are forwarded despite the loss of packets ordered earlier.
- *Lazy-Selective-Retransmits:* nacks are sent for specific lost packets after a short delay to avoid requesting data which was not lost but merely arrived out of order or is sequenced after lost data.
- *Rate-based flow control:* a rate based flow regulator provides explicit support for high delay-bandwidth networks. In addition, the rate based regulator can utilize bandwidth reservations services if such exist in the physical network.

The Hop protocol establishes a bidirectional connection between every two daemons that are directly connected on the overlay network. These two daemons maintain a list of counters and a table of open packets which have not yet been acknowledged. To establish reliable transmission in the presence of losses, Hop uses selective nacks where the receiver requests specific data packets (identified by their sequence number) when loss is detected. The receiver continues to request lost packets until they are recovered. If a lost packet has not been received after N nacks, the Hop protocol declares the link failed and the membership protocol reconfigures the system. This is necessary to eliminate the "failure to receive" problem that can occur either because

Sender handling received NACK:

```
receive NACK n:
  increment_numnacks_received(n)
  resend( get_packet(n) )
  if (get_numnacks_received(n) > MaxNACKS)
     Membership_Link_Failed()
```

Figure 2. Sender handling received NACK

Sender handling ACK Timeout:

```
timeout ACK_Timer:
  ack_val = get_highest_seq_sent_sofar()
  queue_send(ACK ack_val)
  queue_timeout(ACK_Timer)
```

Figure 3. Sender handling ACK Timeout

of a networking fault that deletes certain packets or a malicious attacker who keeps removing one particular packet from the network.

The receiver has two methods to detect loss. First, when the receiver receives a packet which is sequenced beyond the next expected packet, it adds the sequence numbers between the highest previously received sequence number and the just arrived packet to a list of proposed lost packets. This method is shown in Figure 4. The receiver schedules a nack packet containing the newly lost sequence numbers to be sent in a short time if the packets do not arrive meanwhile. In real-life experiments we found this delay could avoid false positive losses of about one percent while still requesting truly lost packets early enough to receive the retransmission and acknowledge it before hitting the maximum outstanding packet limit. Currently this delay is fixed, but we are investigating the amount of reordering of packets on wide area networks and how to set the delay based on the network characteristics.

Second, to detect loss when no further packets are sent, or when there is a long time before the next packet, the sender sends a link acknowledgement to the receiver periodically (based on time and number of packets sent and received), which specifies the highest sequence value the sender has sent. This is shown in Figure 3. Sequence numbers that are equal to or below the specified value that the receiver has not yet received, are added to the list of missing packets. After a short delay these sequence numbers are sent to the sender in a nack packet.

When the sender receives a nack packet it adds the packets represented by the requested sequence numbers to its outgoing retransmit queue as shown in Figure 2. The packets will be sent along with other data when flow control allows. Retransmissions are sent even when the limit on the number of outstanding packets has been reached. Therefore the packet will cross the link in a bounded time or else the link will be declared failed.

The Hop protocol eliminates duplicates by checking every received packet against the list of known missing packets. If the received packet's sequence number is less than the highest previously received either it is in the list of missing packets or it has already been received, so if it is not in the list it is a duplicate and is discarded.

To enable the sender to release buffered copies of packets, the receiver sends link acknowledgements either periodically or after some number of packets, whichever is sooner. The generation of these acks can be seen in the second part of Figure 4. These acknowledgements contain the sequence number of the packet for which all previous packets were received. This combination of acks and nacks provides a fully reliable channel with bounded buffers. However, it does not create any restrictions on the ordering of messages so that each packet is delivered to the higher layers of Spread as soon as it is received by the Hop protocol.

The Hop protocol uses rate-based flow control to limit the rate packets are sent, and a maximum window of outstanding packets to provide termination guarantees for the reliability protocol as described above. The rate regulator is a leaky bucket with both a maximum burst size and an average rate limitation. All of these parameters are set per link so that they can be tuned separately for each link in the overlay network.

Figure 5 presents a case with three hop links where data flows from site A to sites B, C, and D. Messages are assigned different sequence numbers on different links according to their arrival at the parent node. The message labels m1-m4 represent the message identifier and not the sequence number assigned on each link. Suppose, as shown in Figure 5.(a), that packet m2 is lost on the link between sites A and B and subsequently packet m1 is lost on the link between B and C. Note that the loss of packet m2 does not preclude B from forwarding packet m3. Figure 5.(b) shows the nacks for messages m1 and m2 and the concurrent forwarding of m4. In Figure 5.(c) message m2 is recovered by B and immediately forwarded to C and D. Then message m1 is retransmitted to C. Note, that to allow this aggressive behavior, the order for the sequence numbers between sites B and C were as follows: m1 got link sequence 1, m3 got link sequence 2, m4 got link sequence 3, and m2 got link sequence 4. This is the reason a request for m2 was not triggered by site C.

In Section 4 we evaluate both the Hop protocol and TCP for their usefulness in providing a link protocol for the Spread group communication system.

Receiver handling PACKET:

```
receive PACKET n:
  if (n > next_expected )
     add_to_missing_list(next_expected + 1 .. n)
     next_expected = n + 1
     queue_send(NACK next_expected + 1 .. n, Nack_Delay)
  if (n == next_expected )
     deliver(PACKET n )
     next_expected = n + 1
  if (n < next_expected )
     if (check_in_missing_list(n) )
        remove_from_missing_list(n)
        deliver(PACKET n)
     else
        discard(PACKET n)
```

Receiver handling PACKET for ACK:

```
receive PACKET n:
  increment_packets_since_last_ack()
  if (get_packets_since_last_ack() > MaxPacketsBetweenAck)
     ack_val = get_highest_sequence_all_received_upto()
     queue_send(ACK ack_val)
     reset_packets_since_last_ack()
```

Figure 4. Receiver handling PACKET

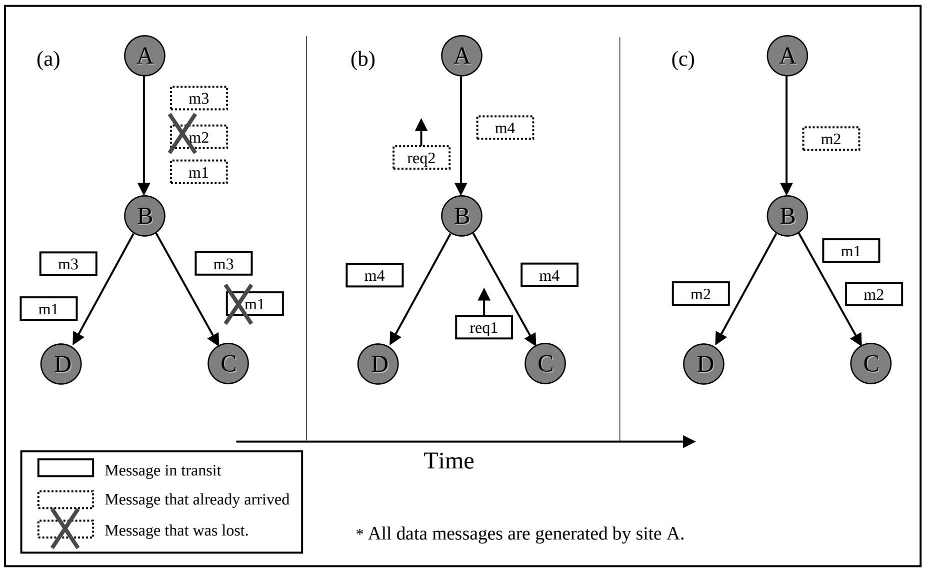

Figure 5. Hop Protocol Forwarding Scenario

4 Performance and Results

4.1 Experimental Setup

We have conducted experiments over the Internet to test the correctness of the implementation and to measure the performance of the different protocols in practice. Figure 1 presents the layout of our testbed which consisted of six sites:

- Hopkins - at the Computer Science department at Johns Hopkins University, Maryland.
- CNDS - our lab at the Center for Networking and Distributed Systems at Johns Hopkins.
- UCSB - at the ECE department at the University of California, Santa Barbara.
- Mae East - on AboveNet Communications network at one of the Internet main connecting hubs, Virginia.
- OSU - at the math department at Ohio State University.
- Rutgers - at the Center for Information Management, Integration and Connectivity at Rutgers University, New Jersey.

Since the focus of this paper is architectural support and protocols for the wide area setting, only one computer from each site participated in the experiments. The computers involved ranged from Sparc-5 to Ultra-5 workstations running Solaris, and Pentium II workstations running Linux. During the tests the computers were also under normal user load, and no changes were made to their operating system. Since none of the Spread protocols use wall clock time, no effort was made to synchronize the system clocks of the machines.

The network characteristics of any particular connection over the Internet may vary significantly depending on the time of day, other users, news events, etc. To minimize the effects of these variations on our experiments, each experiment was conducted a number of times (between 30 to 200 times depending on the specific experiment). Each set of measurements for one experiment was run at approximately the same time (within a few minutes of each other). Separate experiments (reported in separate graphs or tables) were run at different times and so comparing results between tables might not be highly accurate. Because of these variances, any one data point may vary over time, however we believe the aggregate trends of the graphs and results are valid.

4.2 Overlay Networks

For this experiment we created two different overlay networks between the above six sites by adjusting the weights of the links in the configuration of Spread. A Fanout network contains a direct link between each two sites so that every source sends directly to every other site. This was created by assigning equal weights to every link. A shared-tree multicast network was created as shown in Figure 1. This tree was constructed based on measurements of network latency. The experiment shown in Table 1 was conducted using TCP as the link protocol in Spread.

Table 1. Throughput using TCP and several network configurations.

Sending Site	Fanout	Tree
Mae East (Kbits/sec)	487.32	559.79
CNDS (Kbits/sec)	666.94	560.71
Rutgers (Kbits/sec)	184.99	568.34
UCSB (Kbits/sec)	328.81	578.84

In tests run on each of the two networks (Fanout and Tree), for every test, one of the four sites (Mae East, CNDS, Rutgers, UCSB) was a source of a stream of 10000 reliable messages of 1024 bytes. The sending application on that site always made messages available to Spread. The remaining five sites were running a receiving application that computed the running time of the test at that site. The numbers in Table 1 represent the throughput of the slowest receiving site measured in kilobits per second. The difference between the fastest and slowest receiver in most of the tests was negligible. As in any reliable multicast system, the maximum sustained throughput is limited to the throughput of the slowest link.

Table 1 shows how both fanout and multicast tree overlay networks are each better than the other for different source sites. When the CNDS site is the source, the fanout network provides better throughput. This is probably because CNDS has extremely high throughput connectivity to the Internet and thus the first few hops do not form a bottleneck. However, when Rutgers or UCSB are multicasting, the multicast tree network yields much better throughput. Even though the Mae East site is located very close to a major Internet backbone peering point, providing better connectivity then almost any typical server, the multicast tree network was still 15 percent better then the fanout network.

This experiment validates the usefulness, described in Section 2, of source based routing using the overlay networks. For example, while messages generated by CNDS can be sent through a fanout configuration, messages sent by UCSB will be sent using a tree configuration.

4.3 Link Protocols

Here we evaluate the tradeoffs of using the Hop protocol versus using a TCP based link protocol. We started by evaluating the overhead latency associated with each protocol on one link. Then, the latency on a multi-link network was evaluated with regards to number of links, the size of the packets, and the load on the network. Finally, the throughput of a link and a full network were evaluated under varying levels of additional packet loss.

All latency tests were done by an application level program which multicasts a reliable Spread message to a group, and then listens for a response message. A second application runs on the other site and acts as an echo-response server, sending anything it receives immediately back to the sender through Spread. The sender application calculates round-trip latency times by taking the difference between the time it received the echo-response and the time it sent the original message. These latency tests are repeated 30 times back to back and the minimum, average, and maximum are reported. All results are reported as round-trip times, which include time transferring the message from the client to the Spread daemon, processing time in the daemon, network transfer time, the receiving daemon's processing time and the transfer to the receiving application, and a similar reverse path back to the sender. For the tables and figures reporting 'ping' results, the standard 'ping' program was run from between the daemons using 1024 byte packets. We believe the ping latencies provide us with an effective lower bound.

Table 2. Link Latency (Mae East to UCSB).

	ping	tcp	hop
min (ms)	103.3	108.946	107.798
average (ms)	104.1	136.277	108.493
max (ms)	106.8	311.649	110.944

Table 2 shows the single link latency for a link between Mae East and UCSB for 1024 byte messages. Clearly the ping latency is the best, however both the TCP link protocol and the Hop link protocol have minimum times very close to ping. The Hop protocol also is very stable across all the tests, with a variance of only 3 milliseconds, the same as ping, while TCP produced a large variance of over 200 milliseconds between the minimum and maximum latency.

To more realistically evaluate latency over a wide-area network, we also constructed an overlay network consisting of six sites in a chain. This chain is shown in Figure 1, as running from Mae East to UCSB to OSU to Rutgers to CNDS to Hopkins. Note, we realize this is not a practical setup, or even an efficient chain. However, using this chain demonstrates how the protocols interact when packets must be forwarded many times, and how the performance of the protocols scales with the diameter of the multicast network.

The experiments reported in Figure 6 and Figure 8 use the chain network. The sender application is always run from one of the ends of the chain. The receiver application is placed on each of the other sites, and 30 to 200 latency tests each using a 1024 byte reliable message are run. The results of the tests are averaged and graphed. The ping line on the graph was calculated by adding the individual ping times from site to site along the chain.

The results in Figure 6 show how the Hop latency stays close to the ping latency as the number of hops and distance traveled increases, while the TCP latency is significantly higher. This is made more clear in Figure 7 which graphs the percentage overhead of TCP and Hop in comparison with ping times. TCP has an overhead of between 38 and 66 percent on all number of links, while Hop has an overhead of at most 18 percent and as little as 5 percent.

Figure 8 shows the same chain network with the sender placed at Hopkins instead of Mae East. Here the improved end-to-end latency of Hop over TCP as the network latency increases becomes clearer. When the network ping latency increases significantly after OSU, the TCP latency increases even more, while Hop latency stays within a small percentage of ping latency. Figure 9 shows how the percentage overhead of Hop decreases substantially as the network latency increases. When the network latency is small, for example on the local area networks connecting CNDS and Hopkins, the application and IPC overhead of Spread become comparable with the actual network latency. In a working Spread configuration, these local area, low latency networks would use the Ring protocol instead of TCP or Hop. The Ring protocol is designed for local area networks and has excellent performance.

Next, we evaluate the latency for various packet sizes. The results presented in Figure 10 use the same chain network with Mae East as the sender and Hopkins as the only receiver. In this test, reliable messages of varying sizes were sent by the application. The larger message sizes are actually sent as several packets on the physical network, however since each message was only sent after the previous one was received this does not become a throughput test. The Hop protocol does not have a large increase in latency beyond what can be attributed to the size of the message in comparison with TCP, which has a significant increase in latency for packets above 1024 byte.

Next, we evaluate the latency under load. The results presented in Figure 11 use the same chain network. In this test a load application using Spread was flooding the net-

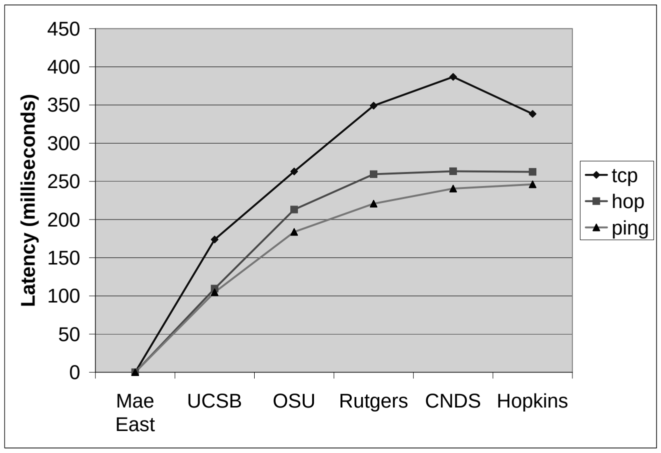

Figure 6. Chain Latency (Mae East)

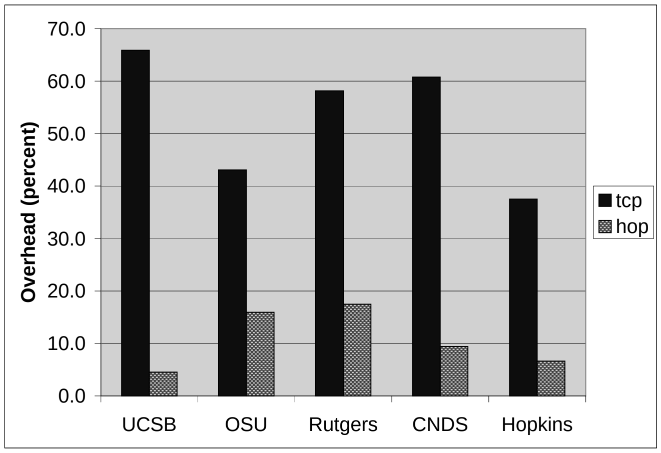

Figure 7. Protocol Latency Overhead (Mae East)

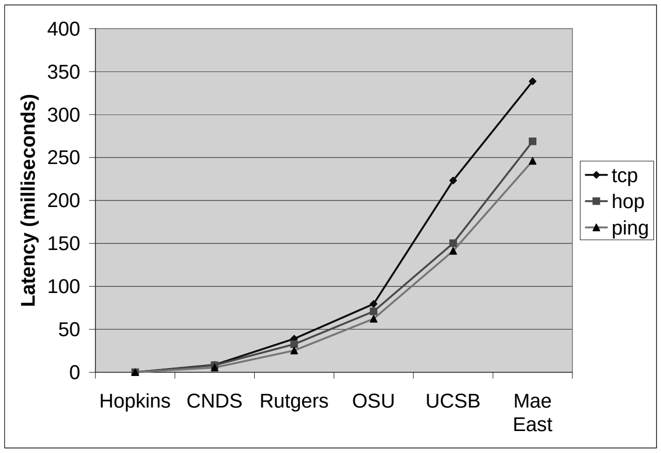

Figure 8. Chain Latency (Hopkins)

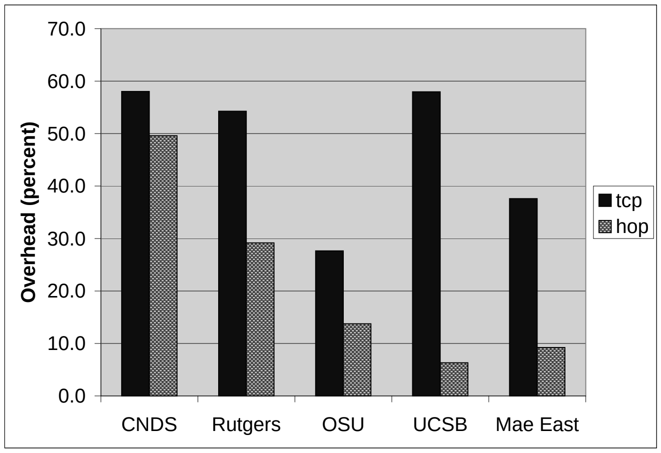

Figure 9. Protocol Latency Overhead (Hopkins)

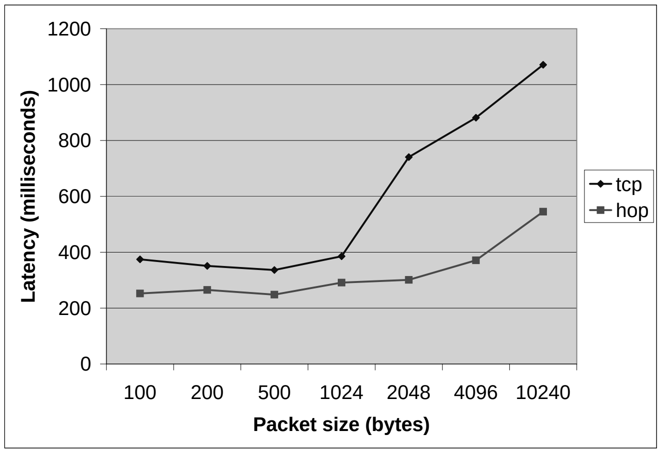

Figure 10. Latency for different size of packets

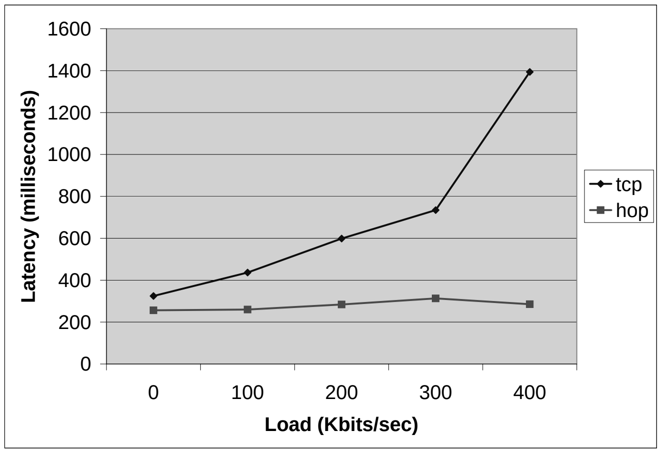

Figure 11. Latency under Load

Table 3. Link Throughput under loss.

Additional Loss Rate	0%	5%	10%	20%
Throughput (Kbits/Sec)	1378.807	1269.043	1113.887	995.000
Throughput (Percentage)	100.0	92.0	80.8	72.2

Table 4. Network Throughput under loss.

Additional Loss Rate	0%	5%	10%	20%
Throughput (Kbits/Sec)	2202.78	2090.45	1807.49	1388.51
Throughput (Percentage)	100.0	94.9	82.1	63.0

work from Mae East with a controlled level of messages per second. Concurrently, the latency test application measured reliable, 1024 byte message latency between Mae East and Hopkins. The Hop protocol has almost constant latency as the background load increases from 0 to 400 kilobits per second. The stability under load is attributed to the Hop protocol's forwarding policy, which does not delay packets even when there is loss or other application traffic. TCP latency shows a steady increase as the background load increases with a jump between 300 and 400 kilobits per second where the latency grows to almost a second and a half.

Finally, we evaluated the Hop protocol's behavior under various levels of packet loss. These tests were done on both a single link between Mae East and UCSB, which are shown in Table 3, and on the multicast tree shown in Figure 1, whose results are reported in Table 4. These tests were done by dropping packets randomly based on a uniform distribution on each side of every link. These losses were in addition to any actual packet loss which occurred on the network. All tests were done with a stream of 10000 reliable Spread messages of 1024 bytes by the same testing application as was used in Section 4.2.

In the link experiment, the throughput decreases at 3 to 10 percent more then the actual loss rate. This seems to us quite reasonable. In the network experiment using 6 sites and 5 links the system still maintained a 63 percent throughput even with 20 percent loss on every link. Overall, the degradation on the whole network is less then double the loss rate on a single link.

We believe that the performance demonstrated by the above experiments validate the viability and usefulness of the Hop protocol in real-life system settings.

5 Related Work

Group communication systems in the LAN environment have a well developed history beginning with ISIS [7], and more recent systems such as Transis [3], Horus [18], Totem [4], and RMP [21]. These systems explored several different models of Group Communication such as Virtual Synchrony [6] and Extended Virtual Synchrony [16]. Newer work in this area focuses on scaling group membership to wide-area networks [5].

A few of these systems have added some type of support for either wide-area group communication or multi-LAN group communication. The Hybrid paper [19] discusses the difficulties of extending LAN oriented protocols to the more dynamic and costly wide-area setting. The Hybrid system has each group communication application switch between a token based and symmetric vector based ordering algorithm depending on the communication latency between the applications. While their system provides a total order using whichever protocol is more efficient for each participant, Hybrid does not handle partitions in the network, or provide support for orderings other then total.

The Multiple-Ring Totem protocol [2] allows several rings to be interconnected by gateway nodes that forward packets to other rings. This system provides a substantial performance boost compared to a single-ring on large LAN environments, but keeps the assumptions of low loss rates and latency and a fairly similar bandwidth between all nodes that limit its applicability to wide-area networks. The latency of the Totem multiple ring protocol has been theoretically analyzed in [20].

The Transis wide-area protocols Pivots and Xports by Nabil Huleihel [14] provide ordering and delivery guarantees in a partitionable environment. Both protocols are based on a hierarchical model of the network, where each level of the hierarchy is partitioned into small sets of nearby processes, and each set has a static representative who is also a member of the next higher level of the hierarchy. Messages can be contained in any subtree if the sender specifies that subtree. The Congress work [5] approaches the problem of providing wide-area membership services separately from actual multicast and ordering services, and provides a general membership service that can provide different semantic guarantees.

IP-Multicast is actively developed to support Internet wide unreliable multicasting and to scale to millions of users. Many reliable multicast protocols which use IP-multicast have been developed, such as SRM [10], RMTP [15], Local Group Concept (LGC) [13], and HRMP [11].

The development of reliable protocols over IP-Multicast has focused on solving scalability problems such as Ack or Nack implosion and bandwidth limits, and providing useful reliability services for multimedia and other isochronous applications. Several of these protocols have developed localized loss recovery protocols. SRM uses randomized timeouts with backoff to request missed data and send repairs, which minimizes duplicates, and has enhancements to localize the recovery by using the TTL field of IP-Multicast to request a lost packet from nearer nodes first, and then expand the request if no one close has it. Several other variations in localized recovery such as using administrative scope and separate multicast groups for recovery, are discussed in [10].

Other reliable multicast protocols like LGC use the distribution tree to localize retransmits to the local group leader who is the root of some subtree of the main tree. RMTP also uses “Designated Receivers” (DR) who act as the head of a virtual subtree to localize recovery of lost packets and provides reliable transport of a file from one sender to multiple receivers located around the world. RMTP is based on the IP-Multicast model, but created user-level multicast through UDP and modified *mrouted* software. RMTP did not examine the tradeoffs in link protocols discussed in this paper because it handles reliability over the entire tree, with the DR’s only acting as aggregators of global protocol information. Since Spread already has additional information for membership and ordering purposes about the exact dissemination of messages and where copies are buffered, Spread can use more precise local recovery to get the packet from the nearest source.

HRMP [11] is a reliable multicast protocol which provides a efficient local reliability based on a ring, while using standard tree-based protocols such as ack trees to provide reliability between rings. This work theoretically analyzes the predicted performance of such a protocol and shows it to be better then protocols utilizing only a ring or a tree. Our work here focuses on the best protocols to use for reliability on a wide-area multicast tree and thus is orthogonal with which local site protocol to use. The Spread system actually uses a ring protocol for local area networks for many of the same reasons HRMP does.

6 Conclusion

We presented an architecture for wide area group communications that was implemented in the Spread system. This architecture takes advantage of the ability to construct user level overlay networks to efficiently disseminate reliable messages to process groups.

We described Hop, an efficient point-to-point reliable transport protocol for connecting sites on a wide area multicast tree. Experiments conducted over the Internet validated the low latency and high stability of the Hop protocol under various load and loss conditions.

Acknowledgements

We would like to thank Michal Miskin-Amir, one of the creators of Spread. We thank Jithesh Parameswaran for programming the optimal routing computations in Spread. We also wish to thank Nabil Adam, Richard Holowczak, Michael Melliar-Smith, Louise Moser, Alec Peterson, and Robert Stanton for allowing us to use their systems in our testings.

References

[1] ACM. Special issue on group communications systems. *Communications of the ACM*, 39(4), April 1996.

[2] D. Agarwal, L. E. Moser, P. M. Melliar-Smith, and R. K. Budhia. The totem multiple-ring ordering and topology maintenance protocol. *ACM Transactions on Computer Systems*, 16(2):93–132, May 1998.

[3] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication subsystem for high-availability. In *Digest of Papers, The 22nd International Symposium on Fault-Tolerant Computing Systems*, pages 76–84, 1992.

[4] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. Agarwal, and P. Ciarfella. The totem single-ring ordering and membership protocol. *ACM Transactions on Computer Systems*, 13(4):311–342, November 1995.

[5] T. Anker, G. V. Chockler, D. Dolev, and I. Keidar. Scalable group membership services for novel applications. In M. Mavronicolas, M. Merritt, and N. Shavit, editors, *Proceedings of the workshop on Networks in Distributed Computing*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 1998.

[6] K. P. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *11th Annual Symposium on Operating Systems Principles*, pages 123–138, November 1987.

[7] K. P. Birman and R. V. Renesse. *Reliable Distributed Computing with the Isis Toolkit*. IEEE Computer Society Press, March 1994.

[8] A. Fekete, N. Lynch, and A. Shvartsman. Specifying and using a partionable group communication service. In *Proceedings of the 16th annual ACM Symposium on Principles of Distributed Computing*, pages 53–62, August 1997.

[9] R. W. Floyd. Algorithm 97 (shortest path). *Communications of the ACM*, 5(6):345, 1962.

[10] S. Floyd, V. Jacobson, C. Liu, S. McCanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM Transactions on Networking*, 5(6):784–803, December 1997.

[11] L. Gu and J. Garcia-Luna-Aceves. New error recovery structures for reliable networking. In *Proceedings of the Sixth International Conference on Computer Communications and Networking*, September 1997.

[12] K. Guo and L. Rodrigues. Dynamic light-weight groups. In *Proceedings of 17th International Conference on Distributed Computing Systems*, pages 33–42, May 1997.

[13] M. Hofmann. A generic concept for large-scale multicast. In B. Plattner, editor, *International Zurich Seminar on Digital Communications*, number 1044 in Lecture Notes in Computer Science, pages 95–106, Februrary 1996.

[14] N. Huleihel. Efficient ordering of messages in wide area networks. Master's thesis, Institute of Computer Science, The Hebrew University of Jerusalem, Jerusalem, Israel, 1996.

[15] J. Lin and S. Paul. Rmtp: A reliable multicast transport protocol. In *Proceedings of IEEE Infocom*, pages 1414–1424, March 1996.

[16] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended virtual synchrony. In *Proceedings of the IEEE 14th International Conference on Distributed Computing Systems*, pages 56–65, June 1994.

[17] J. Nonnenmacher and E. W. Biersack. Performance modelling of reliable multicast transmission. In *Proceedings of INFOCOM 97*, April 1997.

[18] R. V. Renesse, K. Birman, and S. Maffeis. Horus: A flexible group communication system. *Communications of the ACM*, 39(4):76–83, April 1996.

[19] L. E. Rodrigues, H. Fonseca, and P. Verissimo. A synamic hybrid protocol for total order in large-scale systems. In *Proceedings of the 16th International Conference on Distributed Computing Systems*, May 1996. Selected portions published in.

[20] E. Thomopoulos, L. E. Moser, and P. M. Melliar-Smith. Analyzing the latency of the totem multicast protcols. In *Proceedings of the Sixth International Conference on Computer Communications and Networks*, pages 42–50, September 1997.

[21] B. Whetten, T. Montgomery, and S. Kaplan. A high performance totally ordered multicast protocol. In *Theory and Practice in Distributed Systems, International Workshop*, Lecture Notes in Computer Science, page 938, September 1994.